Power Analysis, What Is Now Possible...

Mehdi-Laurent Akkar[1⋆], Régis Bevan[2], Paul Dischamp[2], and Didier Moyart[2]

[1] Bull CP8, 68 route de Versailles,
78431 Louveciennes, France.
ml.akkar@free.fr
[2] Oberthur Card systems,
25 rue Auguste Blanche, 92800 Puteaux, France.
{r.bevan, p.dischamp, d.moyart}@oberthurcs.com

Abstract. Since Power Analysis on smart-cards was introduced by Paul Kocher [KJJ98], the validity of the model used for smart-cards has not been given much attention. In this paper, we first describe and analyze some different possible models. Then we apply these models to real components and clearly define what can be detected by power analysis (simple, differential, code reverse engineering...). We also study, from a statistical point of view, some new ideas to exploit these models to attack the card by power analysis. Finally we apply these ideas to set up real attacks on cryptographic algorithms or enhance existing ones.

Keywords: Smart-cards, Power analysis, DPA, SPA.

1 Power Consuming Models for Smart-Cards

In cryptographic protocols, we normally assume that the attacker has at most the knowledge of the algorithm used and some input and/or output values. In the attack on smart-cards based on power-analysis, the situation is quite different: we assume the attacker has access to more than this, namely, see (within a certain limit) what is done during the computation. Therefore, we need to specify what kind of knowledge could be extracted from a card.

1.1 Sensitive Instructions

As opposed to modern computers, the smart-card processor is very limited in terms of capabilities, registers and memory. Usually, the arithmetic or the logical operations (xor, or, add...) are executed through a special register (e.g. the A-accumulator for Intel 8051 compatible family). Therefore, the programmer cannot load the variables to registers, execute the instructions on them and afterwards store them in their final positions in the memory. So, during the entire execution of a program, the micro-controller is always loading/saving the output of the calculation to/from memory. Moreover, the limited memory of the

⋆ Research done while at Oberthur Card Systems.

T. Okamoto (Ed.): ASIACRYPT 2000, LNCS 1976, pp. 489–502, 2000.

devices often obliges the programmer to implement his code in a straightforward manner.

The limited set of instructions is another advantage for the attacker: usually, all the instructions involve at most two 8-bits variables for the input and one for the output. In most cases one or two of these variables is a special register. Taking advantage of this, the attacker can easily study nearly all the instruction set.

For these reasons, it seems reasonable to assume that a good point to attack a smart-card is when the processor is loading/saving a value.

1.2 Some Consumption Models

A smart-card, even if it is one of the most simple processors, consists of a lot of different blocks: the processor itself, the memory, the Bus ... This is why we need to consider a very general model to represent it correctly. All these blocks are perturbed by external parameters and so will react differently everytime (the power supply can fluctuate, so the clock ...).

Executing an instruction on a smart-card, like on most micro-controllers, takes several machine cycles. For example, the XOR of two values could be processed as follows by the CPU:

- analyze which instruction is to be executed (e.g. XOR);
- load the variables;
- execute the calculation;
- store the result.

Those operations can of course be pipelined and so are not serial in time.
This first (simple) analysis shows that, to model the power consumption of a smart-card, several things need to be taken into account:

- the instruction which is executed;
- the data involved in the calculation (input, output);
- the location in RAM/ROM of the instruction executed and the data;
- the instructions involved before and after the instruction considered;
- some random fluctuation.

Based on these considerations and some experimentations we consider here the following general model: the power consumption of an instruction I (an array $P[I, 1..n]$ where n is the number of point during the acquisition of the instruction) could be represented as follows:

$$\begin{aligned} P[I] = & P_{gen}[I] \times N_{gen} \\ & + P_{in}[I, V_{in}] \times N_{in} \\ & + P_{out}[I, V_{in}, V_{out}] \times N_{out} \\ & + P_{last}[I', V'] \times N_{last} \end{aligned}$$

Where:

- P_{gen} is the general consumption of the instruction and N_{gen} the fluctuation of P_{gen} (i.e. location of the code in ROM/EEPROM);
- $P_{in}[V_{in}]$ is the power consumption of the input operand (we shall see that in first approximation, the power consumption does not depend on the instruction executed) and N_{in} the noise associated;
- $P_{out}[I, V_{in}, V_{out}]$ is the power consumption of the output operand (this time, of course, it depends on the nature of the instruction) and N_{out} the noise associated;
- $P_{last}[I', V']$ represents the influence of the preceding instructions and their input/output and $N_{last/next}$ the associated noise.

1.3 Consumption Model for the Data

Intrinsic behaviour of the micro-controller hardware shows the following characteristics :

- there are some gates which commute when a value changes ($0 \rightarrow 1$ or $1 \rightarrow 0$);
- the bus drives the bit of information (so the value of a bit changes the consumption);
- some bits influence some other places in the micro-controller (e.g. the carry, the overflow);
- writing a 0 or a 1 does not consume the same power.

Taking these considerations into account, we can define the following different models:

- Global : $P[x] = K_x$
- Linear : $P[x] = \sum_{i=0}^{n_{bits}} x_i \times P_i$
- Flipping: $P[x] = F(x, x_{last})$
- Quadratic: $P[x] = \sum_{i=0}^{n_{bits}} \sum_{j=0}^{n_{bits}} X_{00_{ij}} P_{00_{ij}} + X_{01_{ij}} P_{01_{ij}} + X_{10_{ij}} P_{10_{ij}} + X_{11_{ij}} P_{11_{ij}}$

where:

- $X_{ab_{ij}} = 1$ iff $x_i = a$ and $x_j = b$
- $P_{ab_{ij}}$ is the associated consumption.
- x_i corresponds to bit i of x.

The global model is the most general one: it implies a specific consumption for every possible value.
The linear model assumes that every bit has a specific weight and is independent of the other bits. For example if you fix $P_i = 1$ for all $i \leq n$ you obtain a "hamming weight consumption" model.
The flipping model can represent the case where the last value influences the consumption. For example if you take $F = HW(data \oplus data_{last})$ - where HW is the hamming weight - it represents the number of bit flips between the last data which has been manipulated and the actual one.
The quadratic is more powerful. It can represent component where the consumption depends on the value of the bits taken two at a time.

Now let us see how this model is correlated to reality.

2 Real Smart-Cards

All the results given are based on two different models of smart-card processors[1]: a low cost model and a more powerful one. But after further experimentation it seems that most smart-cards behave in a similar way to the defined consumption models.

We want to point out that some aspects have not been considered in the present paper:

- it was assumed that the instructions all had the same consumption model;
- the influence of past instructions were not taken in account.

2.1 Generalities

It appears that most of the processor consumption is due to the instruction being executed. The associated noise is relatively small and probably takes into account the previous executed instructions; the rest of the consumption comes from the values involved in the calculation (input and output).

Instruction	Data	Last instructions/datas	Noise after filtering
85%	9%	5%	1%

2.2 Validity of the Memory Models in Practice

Many people have concluded (cf. [KJJ98, CJRR99a, BS99]) that the consumption of the card directly depends on the Hamming weight, or the number of changes $0 \leftrightarrow 1$ in the binary value considered. It appears that the Hamming weight model is not adapted to the two smart-cards we studied.

The diagrams in figure 1 show the consumption associated with the storage of a value in RAM ordered by hamming weight, and this for the two different smart-cards. If this hypothesis was true we should see an increasing curve which is not the case. By ordered we mean that the values $x_1 \dots x_{256}$ of the average consumption of the "store-i" instruction were taken and ordered by respecting the following: $x_i < x_j \iff HW(i) < HW(j)$ where $HW(i)$ represents the Hamming weight of i. If $HW(i) = HW(j)$ then we did order the value comparing their consumption.

One can see from figure 1 that for a given consumption, one would get different hamming weight values.

We need to find a more appropriate model. We have ordered the values of consumption to have a reference to the correct model. The goal was to find an order on [0..255] which is close to the real order obtained by sorting the power consumptions. Due to the architecture of the micro-controller we have decomposed the value $n \in [0, 256]$ in its binary form $n_1, .., n_8$. The first idea was to find a "linear" order: this means assigning some weight to the different

[1] the exact models are obviously not given here

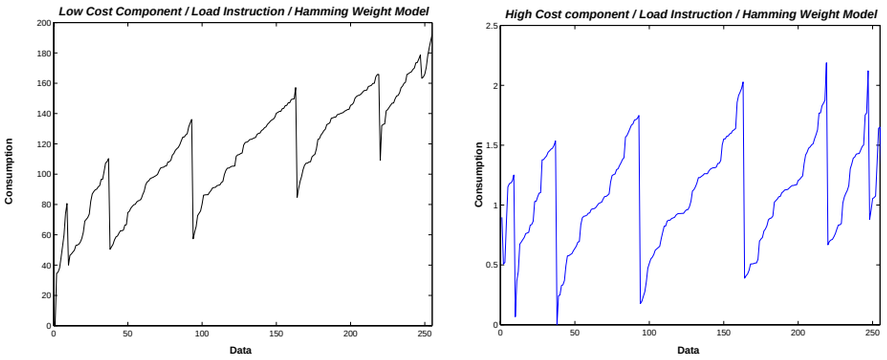

Fig. 1. Hamming Weight Model

bits of n. To compare n to m, once the weights $p_1, ..., p_8$ have been chosen, you compare $n_1.p_1. + ... + n_8.p_8$ to $m_1.p_1 + ...m_8.p_8$. For example the hamming weight corresponds to $p_1 = p_2 = ... = p_8 = 1$ and the natural order to $p_i = 2^{8-i}$. Taking for p_i the average difference of consumption of the bit i (comparing the consumption between "$load(x)$" and "$load(y)$" where $x_j = y_j$ except for the bit i (see below figure 3)) we obtain the following curves (figure 2)

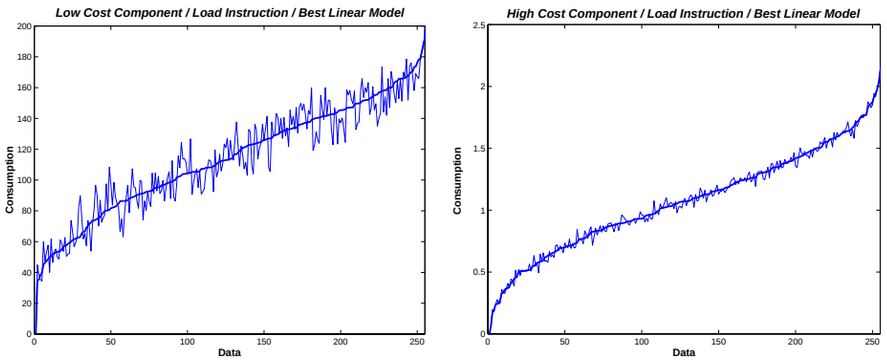

Fig. 2. Best Linear Model

We can clearly see that even if the results are satisfying, we are still far from the expected curve for the low end component (we add a "reference" curve : the curve obtained by sorting the consumption). To check if the "linear" model was correct, we computed for all values the difference caused by changing just one bit in the operand.

The table in fig. 3 presents all the results. The 'bit i' column is obtained by the following computation: the 128 values where the i'th bit was 0 were taken and for each individual value the i'th bit was turned on. The individual consumption changes were measured. Then, we extract the minimum and the maximum of the 128 values and compute the average and the standard deviation to sum up the results:

Component	*Consumption*	*bit* 0	*bit* 1	*bit* 2	*bit* 3	*bit* 4	*bit* 5	*bit* 6	*bit* 7
low cost	*average*	11.9	11.3	11.4	12.1	7.0	32.3	34.5	44.85
	min	−12.1	−25.6	−15.1	−15.4	−30.0	1.2	−8.0	7.2
	max	40.5	42.2	45.7	60.0	45.0	67.6	73.7	86.7
	standard deviation	10.3	11.7	10.8	13.6	14.1	11.9	16.9	16.3
high cost	*average*	12.1	−21.9	16.1	10.7	14.1	−20.1	−5.0	10.3
	min	5.3	−27.7	8.5	5.7	7.7	−25.3	−11.0	3.6
	max	17.0	−15.5	24.0	18.1	19.7	−14.0	2.4	16.9
	standard deviation	2.3	2.4	2.4	2.4	2.4	2.4	2.8	2.7

Fig. 3. Comparison in mV of the influence of one bit on low and high cost component

We can notice some interesting things that explain previous curves:

In both components, one can notice that flipping a bit from 0 to 1 does not affect the consumption by an equal difference. Moreover it can either increase or decrease the consumption. So it shows that in this case the hamming weight model is totally inadequate.

The influence of each bit is not increasing or decreasing with its position inside the byte.

For the low cost component, the variance is very important (i.e. for the bit 6, it can increase or decrease the consumption) explaining that the "best linear model" is not adequate. But, if we consider the following simplified quadratic model: we consider only the terms $b_i b_{i+1}$ (a bit only influences the bit just before and just after); then the curve is very close to the sorted one (not presented here).

For the high cost component the variance is significantly reduced. The silicon founder may have tried to separate the consumption of each bit, inducing this difference.

Now that we have a better model, let us see statistically how much information we can get from a card.

3 Attacks

In this section, we will consider that the following models are good enough to be as precise as the ideal one:

- simplified boolean polynomials of degree 2 for the low cost smart-card;
- best linear model for the high cost smartcard.

3.1 Existing Attacks

One of the most common attack by power-analysis is the DPA attack against DES. For more details, see [KJJ98]. The interesting aspect here is that this attack is based on the following assumption:

On average, the consumption of the card computing with values x such as $x_i = 0$ [2] can be distinguished from the consumption where $x_i = 1$.

We need to prove this assumption. It would be obvious if the card was respecting the Hamming weight model, but, judging by our results, this is not clear. Table 4 presents some results about the efficiency of a DPA attack. We have computed the average consumption of the set of x such as $x_i = 0$ (resp $x_i = 1$).

Component	Consumption for	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
low cost	$x_i = 0$	0.474	0.475	0.475	0.473	0.485	0.429	0.424	0.402
	$x_i = 1$	0.526	0.525	0.525	0.527	0.515	0.571	0.575	0.598
high cost	$x_i = 0$	0.444	0.398	0.425	0.450	0.434	0.406	0.476	0.452
	$x_i = 1$	0.556	0.602	0.575	0.550	0.566	0.593	0.524	0.548

Fig. 4. Efficiency of a DPA attack on low cost and high cost component

The table above confirms the experimental attack: the difference between the two distributions is quite important for every bit of data. But this attack being based on an imperfect model, we have tried to use a better model to enhance existing attacks and imagine new ones.

3.2 PODPA (Perhaps Optimal Differential Power Analysis)

Theory: Ordering the consumption, we compute the sets of value A and B such that:

- A and B are disjoint, $A \cup B = [0, ..255]$
- $|A|=|B|=128$
- $\forall (x, y) \in A \times B$, consumption$(x) <$ consumption(y)

[2] x_i is the bit i of x

Then we obtain a much higher difference between the consumption of subset A and B than in the usual DPA case:

- 36.7% / 63.3% for the low cost component;
- 34.3% / 65.7% for the high cost one.

In real attacks, this results in an improvement by a factor 5 (number of messages needed for a successful attack). Moreover, in critical situations (low quality acquisition, cards allowing few tries), it could result in a sufficient improvement to make an otherwise unsuccessful attack work.

Practical Scenario: we assume for this example that the scheme of DES is known. As in the classical DPA attack, we repeat the following operations on several DES acquisitions with the message M:

- apply IP (DES Initial Permutation) to M;
- apply the expansion permutation to the 32 right bits of $IP(M)$;
- guess 6 bits of the key (output of the first round key scheduling) and XOR it to the corresponding part of the message;
- those 6 bits will be the input of the S-Box table look-up.

In the classical attack we consider one bit of the output of the S-Boxes. But here, we will just separate the set $[0, 63]$ (6 input bits of the S-Boxes) in two optimal subsets by the method explained in the last section. Then, by analyzing all the 6 bits guesses, we can determine the single correct guess. Indeed, by definition of the subsets, it will be the difference curve with the biggest peak.

Remarks:

To realize this attack, we need to know the optimal linear model for the consumption of the smart-card: in reality, this information is not obtained easily[3]. So we have to find an experimental method to determine the consumption model. The easiest method is to implement the "usual" DPA attack to determine a key on one component. Next, to analyze the leaking instruction (in the DPA attack) for every possible input; one can do this knowing the message and the key. And then the results can be applied to every implementation of crypto-algorithms on the same component.

Even if this example is specific, it highlights under which conditions this attack could be applied: one has to go through the algorithm with a known message until a part of the key is involved and mixed with the message. This situation is quite general for secret and public key algorithms. (i.e. AES candidates, RSA, ECC ...)

[3] Such information is considered proprietary by the silicon manufacturers

3.3 BPA: Binary Power Analysis

We will now study another type of attack, the Binary Power Analysis.

Theory: In many secret key algorithms (like DESX, cellular phone algorithms...), the "whitening" technique is used to increase the size of the key without modifying the algorithm used or using multiple encryption. The method used is the following: before (respectively after) the algorithms core, the message (the result) is xored (whitening operation) with a part of the key.

e.g.: whitening a 8-bit message $M_0 \dots M_7$ with a 8-bit key $K = K_0 \dots K_7$

- for(i=0..8) do $M_i = M_i \oplus K_i$
- C = Encrypt(M)
- for(i=0..8) do $C_i = C_i \oplus K_i$

This method, from a power analysis point of view, prevents a DPA-like simulation on the message. Unfortunately, it is quite easy to attack this operation by power analysis and then use the usual attack on the algorithm core. This is based on the fact that if a bit of the key is 0 the associated bit in the message will remain unaltered.

Here are the BPA attack operations :

- Obtain n curves $\{T_k\}_{k=1..n}$ associated with the messages M_k
- for(i=0..8) do (if there are 8 bytes)
  - for(j=0..8) (8 bits / byte)
    - Separate the curves in 2 subsets according to the bit j of byte i of all messages M_k
    - Subtract the average of the 2 subsets
- Process the results to determine the whitening key possibilities

The last step needs more explanations to understand how the separation is done in practice.

Practical Scenario: The graphic 5 presents the results of the analysis of a whitening operation (just one byte).

In this case, the analysis is quite easy. The XOR operation is executed in 3 steps: first the input values are loaded, then the xor is executed, lastly the result is stored. So, when attacking a special bit, on average the consumption will not change when the key bit is 0 and will when the key bit is 1. In this case, we can quite easily recover the key byte by comparing the beginning and the end of the instruction: $K = 00001001$ [4]

One can find more details about the attack in [CJRR99a].

[4] In this case the second peak of each part is representative of the key bit.

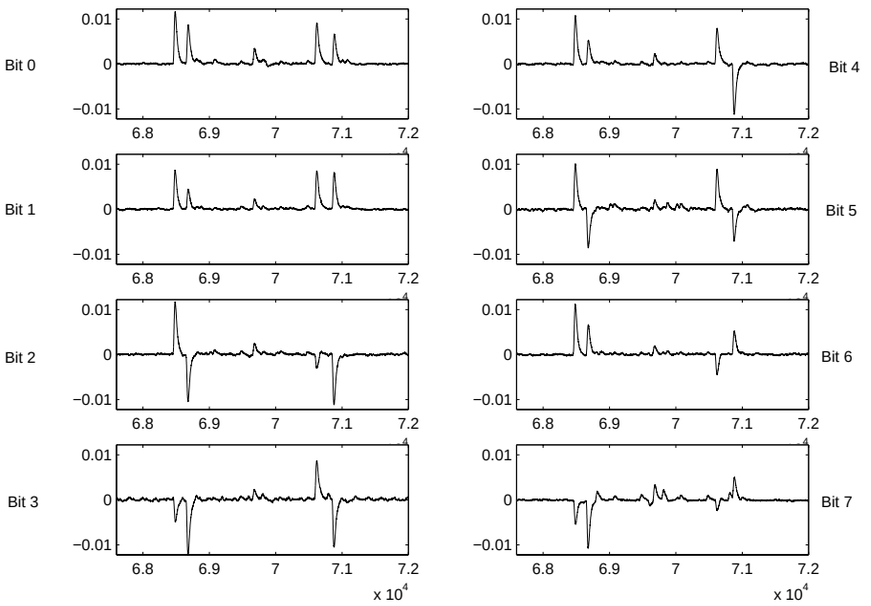

Fig. 5. Whitening operation on a low cost component

3.4 DiPA: Direct Power Analysis

If the consumption model of a card is well known, some very powerful attacks can be set up, avoiding two restriction of DPA like attacks:

- Why just separate the consumption in only two sets?
- Why only do an average differential analysis?

Considering the consumption distribution, it is surely possible to distinguish more than two parts in the consumption curves and build more sophisticated attacks. For examples, one can construct boolean or linear formulas on key-bits or key-bytes like:

- The i-byte of the key is one of these values...
- The bit j and k are equal...
- If bit j and i are different then...
- $b_i+b_j+b_k=2$...

One can then extract some bits of the key by appropriate computation (involving boolean solving methods, Gröbner basis...).

Moreover, if you have a reasonable model for the card, the important point is that all these equations can be obtained with separate acquisitions and does not involve any averaging or differential. You can extract information on each individual computation done by the card.

For example, with the DES, you can extract a lot of information: you can have a very precise idea of the key if you can (and this is possible in reality) localize operations during the end phase of the key-scheduling (just before the xor preceding the S-box operation). You obtain 16 sets of boolean equations on the 6 bits values of the keys. If the accuracy of the model is good (number of subsets in $[0, 255] > 8$) you can recover the entire key with just one acquisition and even without knowing which message is being encrypted.

3.5 Countermeasures

In reality, many countermeasures exist to prevent power analysis attacks on the smart-card:

- desynchronisation of the measures: dummy operations, random frequency clock...;
- randomization of the operations: i.e. random permutations;
- transformations of the data: i.e. public key blinding, Duplication method [GP99], masking methods[Mes00, CJRR99b].

3.6 Using Previous Results

We summarize some generals "counter-countermeasures" to these countermeasures:

- Pattern Recognition (to localize interesting instructions from dummy operations): it appears that the general power comsumption of an algorithm does not change from one acquisition to the other. Moreover it could be that every type of instruction (arithmetic, boolean, load and store ...) and every type of adressing mode has a particular consumption profile. Hence, it might be possible to classify every assembly instruction by its power consumption. Such a study has been undertaken with some success. Knowing the instruction profile, one could disassemble part of the code or at least retrieve some instruction class. Consumption profile helps in synchronising the curves when an external glitch is not provided. Last but not least, an attacker could see what type of countermeasures have been implemented in the code (use of random parameters). The signals are first filtered in order to remove any local noise. Then, the adequate algorithms (classical matching algorithms) are used to retrieve a given instruction in a set of power consumption curves.

- Synchronizing a randomized clock: a hardware counter-measure against waveform attacks consists in using a random clock. Experimentally, it is possible to quite quickly rebuild the synchronous signals from the randomized signals. Once again, the signals are processed with an adequate filter in order to remove any local variations. As a result, it is easy to recognize (studying the first

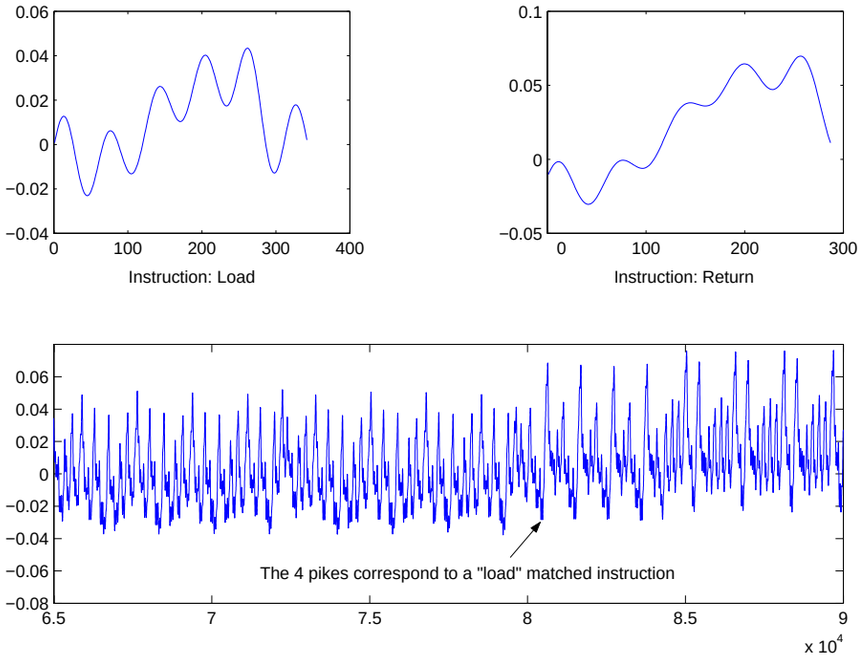

Fig. 6. Pattern Recognition example

and second derivate) the clock cycles. Then, according (or not) to a reference curve, one can expand or shrink every single clock cycle of a given curve. The procedure is then repeated for every clock cycle of the signal. One can then re-use normal DPA attacks.

- Transformations/Masking Methods: we will just present here an example. Often in the DES (due to its structure), the card computes randomly with the data or with the flipped data ($M = M \oplus 0xFF..F$) to protect the card against a bit prediction of the data. But using the PODPA method, we can neutralize this countermeasure. Knowing the consumption distribution of the component, we proceed as follows:

- construct the 128 pairs $(x, x \oplus 255)$ for $x < 128$ with its average consumption $(c(x) + c(x \oplus 255))/2$;
- order the pairs by consumption;
- construct two subsets with the 64 lower/higher consumption pairs;
- proceed a PODPA attack with this distribution.

This attack will work because it does not distinguish x and $x \oplus 255$ for all $x \in [0, 255]$ (there are in the same subset). Moreover, in most components, the consumption of these two subsets is quite different (similar to the usual DPA bit

selection subsets). For example, this attack is very efficient against the countermeasure explained in [Mes00]. It exploits a small leakage when transforming an "arithmetic" mask into a "logical" mask. However we want to point out the fact that the PODPA attack does not defeat "arithmetic" or "logical" masks, just the transformation from one to the other.

Unfortunately, DiPA attacks can counteract masking countermeasures mentioned in ([Mes00, CJRR99b]). The reason is the following: in their security proof, they are assuming that the information is extracted from several messages. But, with our attack some information about the key is extracted from each computation and not from the comparison of different acquisitions.

4 Summary of an Effective Attack

- Acquire sufficient signals to obtain general information.
- Apply adequate DSP[5] to determine the type of clock and retrieve the structure of the program (rounds, countermeasures ...).
- By statistical test (variance) check if the execution is deterministic.
- Obtain sufficient curves of a specific part of the algorithm.
- Rescale if needed (noise, random clock...).
- By usual methods extract one key.
- Use SPA analysis to obtain more information about the card consumption.
- Now one is able to use PODPA, POSPA or DiPA attacks to "easily" break the card.
- One can directly attack other algorithms using the same card model!

5 Conclusions

We have shown that there are several potential attack scenarios which need to be further explored. These attacks require a more detailed study of the component than a classical DPA. A better knowledge of the behaviour of the chip enables to conduct powerful attacks even with little knowledge of the algorithm implementation.

References

[BS99] E. Biham and A. Shamir. Power analysis of the key scheduling of the AES candidates. *Second AES Candidate Conference*, 1999.

[CJRR99a] S. Chari, C. Jutla, J.R. Rao, and P. Rohatgi. A cautionary note regarding evaluation of AES candidates on smart-cards. *CHES*, 1999.

[CJRR99b] S. Chari, C. Jutla, J.R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. *Crypto*, 1999.

[GP99] L. Goubin and J. Patarin. DES and differential power analysis, the duplication method. *CHES*, 1999.

[5] Digital Signal Processing

[KJJ98] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. *Web Site: www.cryptography.com/dpa*, 1998.

[Mes00] T.S. Messerges. Securing the AES finalists against power analysis attacks. *FSE*, 2000.